第 2-8 讲：片内 EEPROM 读写

1. 学习目的

1. 了解 STC8A8K64D4 片内 EEPROM 的分布和特点。
2. 掌握 STC8A8K64D4 片内 EEPROM 分配以及读、写和擦除。

2. 片内 EEPROM 概述

开发产品的时候，我们经常会遇到需要保存数据的应用场景，如一些重要的记录信息或软/硬件配置信息等。这些保存的数据在掉电的情况下是不能丢失的，并且，在需要的时候，用户也可以对数据进行更改。

STC8A8K64D4 系列单片机考虑到了这样的应用场景，他利用 ISP/IAP 技术可将内部 Data Flash 当 EEPROM，从而实现数据的掉电不丢失存储，EEPROM 按照扇区组织的，每个扇区的大小是 512 字节，支持的擦写次数在 10 万次以上。

1. 片内 EEPROM 分布

STC8A8K64D4 系列单片机片内集成了 64K 字节的 Flash（EEPROM），Flash 和 EEPROM 的大小取决于具体的单片机型号，如下表所示。

不同型号的单片机的 Flash 和 EEPROM 大小不同，但是他们的和都是 64K 字节，该系列单片机中，STC8A8K64D4 比较特殊，他的 EEPROM 大小是可以由用户设定的。

表 1：STC8A8K64D4 系列单片机片内 Flash 和 EEPROM 分布

单片机型号	EEPROM 大小	EEPROM 扇区数量
STC8A8K16D4	48K	96
STC8A8K32D4	32K	64
STC8A8K48D4	16K	32
STC8A8K60D4	4K	8
STC8A8K64D4	用户自定义，大小必须是 512 的倍数。	

2. 片内 EEPROM 特点

- STC8A8K64D4 系列单片机片内 EEPROM 的擦除是以扇区为单位进行的。
- EEPROM 是逐字节读写的，写之前，写入地址所在的扇区需要先执行扇区擦除操作。这是因为 EEPROM 的写操作只能将字节中的各个位由 1 写为 0，而不能将 0 写为 1，因此，写之前需要通过扇区擦除操作将整个扇区的位都写为 1。

3. EEPROM 操作时间

- 读取 1 字节：4 个系统时钟（使用 MOVC 指令读取更方便快捷）。
- 编程 1 字节：约 30～40us（实际的编程时间为 6~7.5us，但还需要加上状态转换时间

和各种控制信号的 SETUP 和 HOLD 时间）擦除 1 扇区（ 512 字节）： 约 4～6ms。

EEPROM 操作所需时间是硬件自动控制的，用户只需要正确设置 IAP_TPS 寄存器即可。IAP_TPS＝系统工作频率/1000000（小数部分四舍五入进行取整）。

- 例如： 系统工作频率为 12MHz，则 IAP_TPS 设置为 12。
- 又例如： 系统工作频率为 22.1184MHz，则 IAP_TPS 设置为 22。

4. EEPROM 的访问

EEPROM 的访问方式有两种： IAP 方式和 MOVC 方式。IAP 方式可对 EEPROM 执行读、写、擦除操作，但 MOVC 只能对 EEPROM 进行读操作，而不能进行写和擦除操作。无论是使用 IAP 方式还是使用 MOVC 方式访问 EEPROM，首先都需要设置正确的目标地址。IAP 方式时，目标地址与 EEPROM 实际的物理地址是一致的，均是从地址 0000H 开始访问，但若要使用 MOVC 指令进行读取 EEPROM 数据时，目标地址必须是在 EEPROM 实际的物理地址的基础上还有加上程序大小的偏移。

■ **应用建议：**

- 使用 EEPROM 存储数据时，可以考虑将关联性不强的数据分别存放在不同的扇区。这样，当某类数据需要更新时，就不需要对其他不相关的数据存放扇区进行操作。
- 存储数据时，应对数据更新的频率进行评估，以保证产品的生命周期内擦写次数不超过芯片支持的最大值（10 万次）。如果数据更新频率高，可以考虑将数据在不同的扇区轮换存放，或者在同一个扇区的不同地址轮换存放，以降低使用的 EEPROM 扇区的擦写次数。

3. 软件设计

3.1. EEPROM 读写步骤

EEPROM 操作需要按照下面几个步骤进行，包含规划写入数据的 EEPROM 空间、擦除 EEPROM、写 EEPROM 以及读 EEPROM。

图 1：EEPROM 读写操作步骤

3.1.1. 规划写入数据的 EEPROM 空间

操作 EEPROM 之前，我们需要从片内 Flash 中划分出 EEPROM 空间，STC8A8K64D4

系列单片机中，STC8A8K64D4 型号单片机的 EEPROM 是需要用户自己设定的，其他型号都是已经分配好的，固定大小的。

STC8A8K64D4 型号单片机的 EEPROM 空间划分下载程序时在 STC-ISP 软件中设置的，如下图所示。

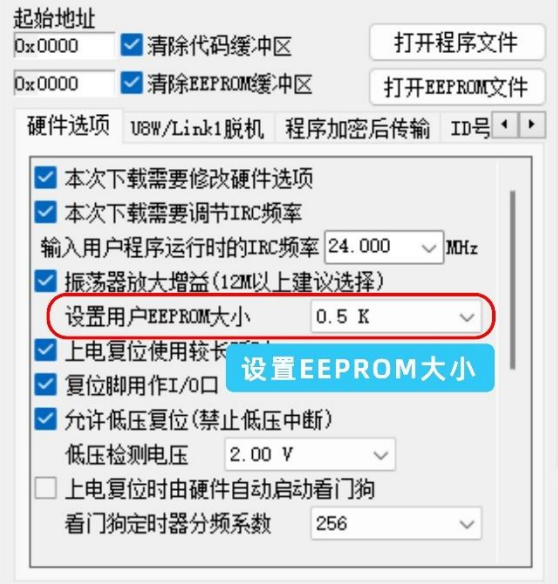

图 2：EEPROM 划分方法

EEPROM 分配总是从顶端向下分配的，下图是 STC8A8K64D4 型号单片机分配 4K 字节 EEPROM 的示例。

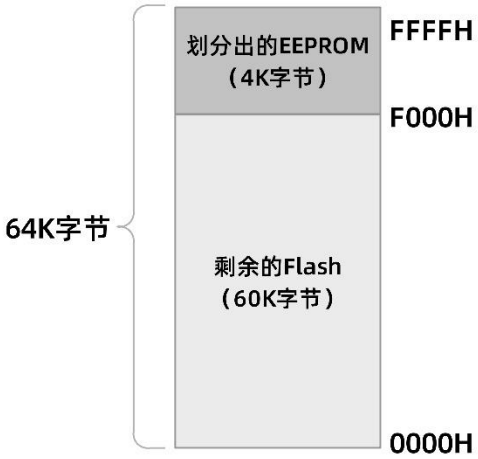

图 3：分配 4K 字节 EEPROM

我们在分配 EEPROM 空间时需要注意以下几点：

- 分配的 EEPROM 空间必须是 512 的倍数，即分配的单位是扇区。
- 分配的 EEPROM 空间不能占用程序存储的空间。

3.1.2. 擦除 EEPROM 扇区

EEPROM 擦除扇区时，只能使用 IAP 方式。擦除流程中需要注意的是，执行扇区擦除

时，输入该扇区的任意地址即可，如擦除 EEPROM 的第一个扇区，地址输入“0x0000~0x01FF”中的任意地址都可以。通常，我们编程时，为了方便指定擦除的扇区，习惯于使用扇区的首地址作为擦除该扇区的地址，如擦除 EEPROM 的第一个扇区，地址输入“0x0000”。

图 4：EEPROM 擦除流程

1. 使能和关闭 IAP

EEPROM 擦除扇区前需要使能 IAP，擦除完成后需要关闭 IAP。使能和关闭 IAP 由“EEPROM 控制寄存器（ IAP_CONTR）”的位7（IAPEN）控制，如下图所示。

EEPROM 控制寄存器（ IAP_CONTR）：

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-

IAPEN：EEPROM操作使能控制位

0：禁止 EEPROM 操作

1：使能 EEPROM 操作

2. 设置等待时间

等待时间由“EEPROM 等待时间控制寄存器（IAP_TPS）”的位5~位0（IAPTPS[5:0]）设置，如下图所示。

EEPROM 等待时间控制寄存器（ IAP_TPS）：

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TPS	F5H	-	-	IAPTPS[5:0]					

IAP_TPS＝系统工作频率/1000000（小数部分四舍五入进行取整）。如本例中使用的系统工作频率为 24MHz，因此 IAP_TPS 设置为 24。

3. 设置 IAP 命令（擦除扇区）

IAP 命令由“EEPROM 命令寄存器（IAP_CMD）”的位1~位0（CMD[1:0]）设置，如下图所示。

EEPROM 命令寄存器（IAP_CMD）：

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CMD	C5H	-	-	-	-	-	-	CMD[1:0]	

- CMD[1:0]：发送 EEPROM 操作命令
- 00： 空操作。
- 01： 读 EEPROM 命令。读取目标地址所在的 1 字节。
- 10： 写 EEPROM 命令。写目标地址所在的 1 字节。 注意：写操作只能将目标字节中的 1 写为 0，而不能将 0 写为 1。一般当目标字节不为 FFH 时，必须先擦除该目标地址所在的扇区。
- 11：擦除 EEPROM。擦除目标地址所在的扇区。注意：擦除操作会一次擦除 1 个扇区（512 字节），整个扇区的内容全部变成 FFH。

4. 设置扇区地址

扇区地址由“EEPROM 地址寄存器（IAP_ADDR）”设置，地址是16位的，IAP_ADDRH 保存扇区地址的高字节， IAP_ADDRL 保存扇区地址的低字节，如下图所示。

EEPROM 地址寄存器（IAP_ADDR）：

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_ADDRH	C3H								
IAP_ADDRL	C4H								

- 扇区地址设置示例：擦除 EEPROM 的第一个扇区（使用该扇区的任意地址都可以，习惯上使用起始地址）

```
IAP_ADDRH = 0xF0;        //地址的高 8 位写入 IAP_ADDRH 寄存器
IAP_ADDRL = 0x00;        //地址的高 8 位写入 IAP_ADDRL 寄存器
```

5. 触发 EEPROM 操作

完成前面的 4 步配置之后，向 EEPROM 触发寄存器（IAP_TRIG）先写入 5AH，再写入 A5H 即可触发扇区擦除。EEPROM 触发寄存器（IAP_TRIG）如下图所示。

EEPROM 触发寄存器（IAP_TRIG）：

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TRIG	C6H								

写完触发命令后， CPU 会处于 IDLE 等待状态，直到相应的 IAP 操作执行完成后 CPU 才会从 IDLE 状态返回正常状态继续执行 CPU 指令。

3.1.3. 写 EEPROM

EEPROM 编程只能使用 IAP 方式。EEPROM 编程步骤如下图所示，他和擦除步骤类似，图中红色字体标注的步骤是和擦除操作不一样的。这里，我们只描述不一样的部分，其他步骤读者参照擦除操作即可。

图 5：EEPROM 写流程

1. 设置 IAP 命令（编程 EEPROM）

编程 EEPROM 时，“EEPROM 命令寄存器（IAP_CMD）”的位 1~位 0（CMD[1:0]）写入数值“10”。

编程时需要注意，待编程的地址所在的 EEPROM 扇区必须是擦除过的，否则，可能会导致错误。

2. 设置 EEPROM 编程地址

和擦除操作地址一样，EEPROM 编程地址也是由“EEPROM 地址寄存器（IAP_ADDR）”设置的。每次编程完成后，IAP_ADDRL 和 EEPROM 命令寄存器 IAP_CMD 的内容不变，即地址不会自动递增，因此，在连续的地址上批量写入数据时，需手动更新地址寄存器 IAP_ADDRH 和寄存器 IAP_ADDRL 的值。

3. 待编程数据写入 IAP_DATA 寄存器

待编程数据在触发 EEPROM 的写操作前，必须写入到“EEPROM 数据寄存器（IAP_DATA）”，每次只能写入一个字节数据。

3.1.4. 读 EEPROM

EEPROM 读取数据可以使用 IAP 方式或 MOVC 方式。IAP 方式读步骤如下图所示，他和擦除、写步骤类似，图中红色字体标注的步骤是和擦除、写操作不一样的。这里，我们只描述不一样的部分，其他步骤读者参照擦除、写操作即可。

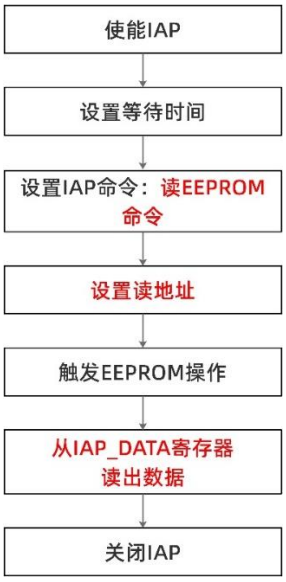

图 6：EEPROM 读流程

1. 设置 IAP 命令（读 EEPROM）

读 EEPROM 时，“EEPROM 命令寄存器（IAP_CMD）”的位 1~位 0（CMD[1:0]）写入数值“01”。

2. 设置读地址

EEPROM 读地址同样是由“EEPROM 地址寄存器（IAP_ADDR）”设置的。每次读完成后，IAP_ADDRL 和 EEPROM 命令寄存器 IAP_CMD 的内容不变，即地址不会自动递增，因此，在连续的地址上批量读出数据时，需手动更新地址寄存器 IAP_ADDRH 和寄存器 IAP_ADDRL 的值。

3. 从 IAP_DATA 寄存器读出数据

读 EEPROM 时，读命令执行完成后读出的 EEPROM 数据保存在 IAP_DATA 寄存器中。程序中，可以从 IAP_DATA 寄存器获取读取的数据。

从 EEPROM 读取数据，还可以使用 MOVC 方式，MOVC 方式读取速度要快于 IAP 方式。使用 MOVC 方式读取时，需要注意地址和 IAP 模式是不一样的，MOVC 方式需要加上偏移地址。如下图所示，当从 Flash 划分 4K 字节的 EEPROM 后，IAP 方式操作 EEPROM 是从地址 0 开始的，MOVC 方式操作 EEPROM 的地址是从 0xF000 开始的。

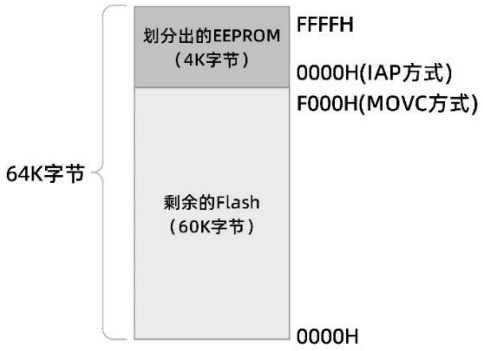

图 7：IAP 方式和 MOVC 方式操作 EEPROM 时的地址

3.2. EEPROM 读写实验

✧ 注：本节的实验是在“实验 2-6-1：串口 1 数据收发实验”的基础上修改，本节对应的实验源码是：“实验 2-8-1：片内 EEPROM 读写”。

3.2.1. 实验内容

本例中，从 Flash 中划分 4K 字节的 EEPROM（共 8 个扇区）。程序中检测 KEY1 按键的状态，当 KEY1 按键按下后，擦除 EEPROM 的第 1 个扇区，然后从该扇区的起始地址开始连续写入 256 字节数据，之后分别通过 IAP 方式和 MOVC 方式读出数据并通过串口输出。

3.2.2. 代码编写

1. 新建一个名称为“eeprom.c”的文件及其头文件“eeprom.h”并保存到工程的“Source”文件夹，并将“eeprom.c”加入到 Keil 工程中的“SOURCE”组。
2. 引用头文件

因为在“main.c”文件中使用了“eeprom.c”文件中的函数，所以需要引用下面的头文件“eeprom.h”。

代码清单：引用头文件

```
1.  //引用 EEPROM 的头文件
2.  #include    "eeprom.h"
```

3. 编写禁止访问 EEPROM 的函数

使用 IAP 方式操作 EEPROM 时，每次操作完成后都需要关闭对 EEPROM 的访问，并清零相关的寄存器，为了方便程序操作，我们编写一个禁止访问 EEPROM 的函数“EEPROM_Disable ()”供 EEPROM 操作调用，代码清单如下。

代码清单：禁止访问 EEPROM

```
1.  /*********************************************************************
2.  * 描  述 : 禁止访问 EEPROM
3.  * 参  数 : 无
4.  * 返回值 : 无
5.  *********************************************************************/
6.  void EEPROM_Disable(void)
7.  {
8.      IAP_CONTR = 0x00;           //禁止 EEPROM 操作
9.      IAP_CMD   = 0x00;           //清零 EEPROM 命令寄存器
10.     IAP_TRIG  = 0x00;           //清零 EEPROM 触发寄存器，防止 ISP/IAP 命令误触发
11.     IAP_ADDRH = 0xFF;         //将地址设置到非 IAP 区域，防止误操作
12.     IAP_ADDRL = 0xFF;
13. }
```

4. 擦除 EEPROM 扇区

因为 EEPROM 的编程原理是只能将各个 bit 由 1 写为 0，而不能将 0 写为 1。因此在 EEPROM 编程之前，为了保证写入的正确性，如果扇区的内容不是 0xFF，则必须将对应

的扇区擦除，使得扇区的内容全部恢复为 0xFF，扇区擦除代码清单如下。

代码清单：擦除指定的 EEPROM 扇区

```
/*************************************************************************
 * 描  述 : 擦除指定的 EEPROM 扇区
 * 参  数 : EE_address: 待擦除扇区的首地址（使用扇区内任意地址都可以，习惯上使用首地址）
 * 返回值 : 无
 *************************************************************************/
void EEPROM_SectorErase(u16 EE_address)
{
    EA = 0;                         //禁止总中断
    IAP_CONTR |= 0x80;              //允许 EEPROM 操作
    IAP_TPS = 24;                   //系统时钟使用的是:24MHz，IAP_TPS=24000000/1000000 = 24
    IAP_CMD = 0x03;                 //写入命令：扇区擦除命令

    IAP_ADDRH = EE_address / 256;   //地址寄存器高 8 位赋值
    IAP_ADDRL = EE_address % 256;   //地址寄存器低 8 位赋值

    IAP_TRIG = 0x5A;                //触发 EEPROM 操作，IAP_TRIG 先写入 0x5A，再写入 0xA5
    IAP_TRIG = 0xA5;
    _nop_();                        //空操作，延时
    EEPROM_Disable();               //禁止访问 EEPROM
    EA = 1;                         //开启总中断
}
```

5. 批量写入数据

批量写入数据实现了以给定的地址作为起始地址连续写入指定长度数据的功能，代码清单如下。注意写入数据时不要超过划分的 EEPROM 的边界。

代码清单：批量写入数据

```
/*************************************************************************
 * 描  述 : 以指定的 EEPROM 地址为起始地址连续写入指定长度的数据
 * 参  数 : EE_address: EEPROM 地址
            p_buf: 指向存放待写入数据的缓存
            w_size: 写入的字节数
 * 返回值 : 无
 *************************************************************************/
void EEPROM_write_bytes(u16 EE_address,u8 *p_buf,u16 w_size)
{
    EA = 0;                     //禁止总中断
    IAP_CONTR |= 0x80;          //允许 EEPROM 操作
    IAP_TPS = 24;               //系统时钟使用的是:24MHz，IAP_TPS=24000000/1000000 = 24
    IAP_CMD = 0x02;             //写入命令：写 EEPROM 命令
    do                          //循环写入数据
    {
```

```
        IAP_ADDRH = EE_address / 256;        //地址寄存器高 8 位赋值
        IAP_ADDRL = EE_address % 256;        //地址寄存器低 8 位赋值
        IAP_DATA = *p_buf;                   //待编程数据写入 EEPROM 数据寄存器
        IAP_TRIG = 0x5A;                     //触发 EEPROM 操作，IAP_TRIG 先写入 0x5A，再写入 0xA5
        IAP_TRIG = 0xA5;
        _nop_();                             //空操作，延时
        EE_address++;
        p_buf++;
    }while(--w_size);
    EEPROM_Disable();              //禁止访问 EEPROM
    EA = 1;                        //开启总中断
}
```

6. 批量读取数据

批量读取数据实现了从以给定的 EEPROM 地址为起始地址连续读出指定长度的数据的功能，读出的数据保存到 p_buf 指向的缓存。

读取数据可以使用 IAP 方式或 MOVC 方式，下面是两种方式的代码清单，读者可以对比一下，不难发现，MOVC 方式比 IAP 方式简单，并且读取速度快于 IAP 方式。

代码清单：批量读取数据：IAP 方式

```
/*************************************************************************
* 描  述 : IAP 方式读取数据。以给定的 EEPROM 地址为起始地址连续读出指定长度的数据，
*        : 读出的数据保存到 p_buf 指向的缓存
* 参  数 : EE_address: EEPROM 地址
*        : p_buf: 指向保存读出数据的缓存
*        : r_size: 读出的字节数
* 返回值 : 无
*************************************************************************/
void EEPROM_read_bytes_iap(u16 EE_address,u8 *p_buf,u16 r_size)
{
    EA = 0;                     //禁止总中断
    IAP_CONTR |= 0x80;          //允许 EEPROM 操作
    IAP_TPS = 24;               //系统时钟使用的是:24MHz，IAP_TPS=24000000/1000000 = 24
    IAP_CMD = 0x01;             //写入 IAP 命令：读 EEPROM
    do
    {
        IAP_ADDRH = EE_address / 256;   //地址寄存器高 8 位赋值
        IAP_ADDRL = EE_address % 256;   //地址寄存器低 8 位赋值
        IAP_TRIG = 0x5A;                //触发 EEPROM 操作，IAP_TRIG 先写入 0x5A，再写入 0xA5
        IAP_TRIG = 0xA5;
        _nop_();                        //空操作，延时
        *p_buf = IAP_DATA;              //读出 EEPROM 数据寄存器的值保存到指定缓存
        EE_address++;                   //地址加 1
        p_buf++;                        //数据缓存地址加 1
```

```
25.     }while(--r_size);
26.     EEPROM_Disable();                   //禁止访问 ISP/IAP
27.     EA = 1;                             //开启总中断
28. }
```

MOVC 方式读取 EEPROM 的代码清单如下：

代码清单：批量读取数据：MOVC 方式

```
1.  /*************************************************************************
2.  * 描  述 : MOVC 方式读取数据。以给定的 EEPROM 地址为起始地址连续读出指定长度的数据，
3.  *        : 读出的数据保存到 p_buf 指向的缓存
4.  * 参  数 : EE_address: 给定的读取数据的起始地址
5.  *        : p_buf: 指向保存读出数据的缓存
6.  *        : r_size: 读出的字节数
7.  * 返回值  : 无
8.  *************************************************************************/
9.  void EEPROM_read_bytes_movc(u16 EE_address,u8 *p_buf,u16 r_size)
10. {
11.     u8 code *p_dat;
12.
13.     p_dat = (u8 *)EE_address;
14.     while(r_size--)       //循环读出数据
15.     {
16.         *p_buf = *p_dat;  //读出 ISP/IAP 数据寄存器的值送往指定缓存
17.         p_dat++;
18.         p_buf++;
19.     }
20. }
```

主函数中，检测到 KEY1 按键按下后，擦除 EEPROM 第一个扇区，接着从该扇区的起始地址开始连续写入 256 字节数据，之后分别通过 IAP 方式和 MOVC 方式读出数据并通过串口输出。

本例中，从 Flash 中划分 4K 字节的 EEPROM（共 8 个扇区）。IAP 方式读取数据时，第一个扇区的起始地址是 0x0000。使用 MOVC 方式读取数据时，第一个扇区的起始地址是 0xF000。程序代码清单如下。

代码清单：EEPROM 读写测试

```
1.  //定义 EEPROM 读写软件缓存，大小各为 256 字节
2.  xdata u8 read_buf[256],write_buf[256];
3.
4.  /*************************************************************************
5.  功能描述：主函数
6.  参    数：无
7.  返 回 值：int 类型
8.  *************************************************************************/
9.  int main(void)
```

```
{
    u8 btn_val,j;
    u16 i,test_len;

    P2M1 &= 0x3F;   P2M0 &= 0x3F;     //设置 P2.6~P2.7 为准双向口（指示灯 D1 和 D2）
    P7M1 &= 0xF9;   P7M0 &= 0xF9;     //设置 P7.1~P7.2 为准双向口（指示灯 D4 和 D3）
    P3M1 &= 0xFE;   P3M0 &= 0xFE;     //设置 P3.0 为准双向口（串口 1 的 RxD）
    P3M1 &= 0xFD;   P3M0 |= 0x02;     //设置 P3.1 为推挽输出（串口 1 的 TxD）

    P3M1 &= 0x3F; P3M0 &= 0x3F;     //设置 P3.6~P3.7 为准双向口（按键 KEY2 和 KEY1）
    P0M1 &= 0x5F;   P0M0 &= 0x5F;   //设置 P0.5，P0.7 为准双向口（按键 KEY4 和 KEY3）

//如果按键电路上没有外部上拉电阻，需要开启 GPIO 的片内上拉。
//开发板的按键电路设计了上拉电阻，因此，无需开启片内上拉
//  P_SW2 |= 0x80;                  //将 EAXFR 位置 1，以访问在 XDATA 区域的扩展 SFR
//  P0PU |= 0xA0;                 //开启 P0.5、P0.7 的上拉电阻
//  P3PU |= 0xC0;                 //开启 P3.6、P3.7 的上拉电阻
//  P_SW2 &= 0x7F;                  //将 EAXFR 位置 0，恢复访问 XRAM

    uart1_init();                   //串口 1 初始化
    EA = 1;                         //使能总中断
    test_len = 256;                 //EEPROM 读写测试长度为 256 个字节
    while(1)
    {
       btn_val=buttons_scan(0);      //获取开发板用户按键检测值，不支持连按
       //按下 KEY1：测试按页写入。扇区 0 的第一页写入 256 个字节，之后读出数据并通过串口输出
       if(btn_val == BUTTON1_PRESSED)
       {
          led_toggle(LED_1);  //翻转指示灯 D1 的状态，指示按键按下
          //写之前需要先执行擦除操作，擦除 EEPROM 第一个扇区
          EEPROM_SectorErase(0x0000);
          //测试数据赋值
          j = 0;
          for(i=0;i<256;i++)write_buf[i] = j++;

          //写入测试数据
          EEPROM_write_bytes(0x0000,write_buf,test_len);
          //读出测试数据：IAP 方式
          //在 EEPROM 的首地址为 0x1000 处读取 5 个字节存入 buffer 数组中
          EEPROM_read_bytes_iap(0x0000,read_buf,test_len);
          //串口打印读取的数据
          for(i=0;i<test_len;i++)printf("%02bX ",read_buf[i]);
          printf("\r\n");
```

```
53.             //读出测试数据：MOVC 方式
54.             EEPROM_read_bytes_movc(0xF000,read_buf,test_len);
55.             //串口打印读取的数据
56.             for(i=0;i<test_len;i++)printf("%02bX ",read_buf[i]);
57.         }
58.     }
59. }
```

3.2.3. 硬件连接

本实验需要使用 LED 指示灯 D1、按键 KEY1 和 USB 转串口，按下下图所示短接对应的跳线帽。

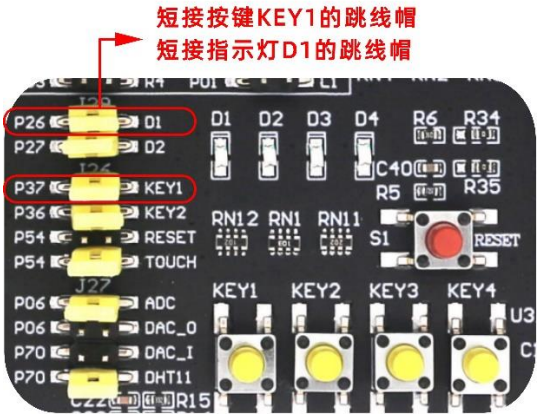

图 8：D1 和 KEY1 跳线帽短接

图 9：USB 转串口跳线帽短接

3.2.4. 实验步骤

1. 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-8-1：片内 EEPROM 读写”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题）。
2. 双击“…\eeprom\Project\object”目录下的工程文件“eeprom.uvproj”。
3. 点击编译按钮编译工程，编译成功后生成的 HEX 文件“eeprom.hex”位于工程的“…\eeprom\project”目录下。
4. 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：24MHz，用户

EEPROM 大小设置为 4K。

5. 电脑上打开串口调试助手，选择开发板对应的串口号，将波特率设置为 9600bps。
6. 程序运行后，按下 KEY1 按键，会擦除 EEPROM 的第一个扇区，接着从第一个扇区起始地址开始连续写入 256 个字节数据，之后分别使用 IAP 方式和 MOVC 方式读出数据并通过串口输出数据。

图 14：串口接收的数据